

REMARKS

The Examiner's action dated March 25, 2004, has been received and its contents carefully noted. In order to advance prosecution, claims 1, 4-7, 9-12, 14, 15, 18-25, 28 and 29 have been amended and dependent claims 36-43 have been added. Claims 1 and 3-43 are now pending.

The rejections under 35 U.S.C. §112, first and second paragraphs are traversed for reasons to be presented below.

35 U.S.C. §112, first paragraph

The Examiner's comments have been carefully reviewed by the Applicant. The independent claims rejected on this ground have been amended to address the Examiner's criticism. Nevertheless, the rejection is traversed for the reason that, even with regard to the previous version of those claims, the rejection under 35 U.S.C. §112, first paragraph is unworkable.

The specific ground of the rejection is "failing to comply with the enablement requirement", while the explanation of the rejection: "There is no indication in Claim 7 and similar Claims when a debugger command is placed in the trace file..." simply has no bearing on enablement.

"The enablement requirement refers to the requirement of 35 U.S.C. §112, first paragraph that the *specification* describe how to make and how to use the claimed invention." MPEP §2164. The present *specification* discloses when a debugger command is placed in the trace file and the Examiner has not asserted otherwise. Since this aspect of the invention has been disclosed in the specification, it must be concluded that the specification is enabling with regard to when a

debugger command is placed in the trace file. It is not the function of claims to disclose how an invention is carried out; rather their function is to define the contribution of an invention over the prior art.

If a claim defines a feature that is not sufficiently disclosed in the specification, there may be an enablement issue; if a feature that is sufficiently disclosed in the specification is not specifically recited in a claim, there is no enablement issue. It is simply a matter of claim breadth.

Applicant respectfully wonders whether perhaps the Examiner has misunderstood the claim since he asks "When is this debugger command placed in the trace file? There is no indication in Claim 7 and similar Claims when a debugger command is placed in the trace file after the trace file is created in step (ii)."

It is respectfully submitted that so far as claim 7 and the other rejected claims are concerned, there is no need to recite the action of embedding the debugger command in the trace file since it is not intended that those claims distinguish over the prior art on the basis of the manner or timing of the embedding operation. Thus, although the Examiner is, of course, correct that chronologically the debugger command is embedded in the trace file after the trace file is created in step (ii), it is not correct to imply that this must find expression in the claim. We remind the Examiner that Claim 7 covers two possibilities: one is that the trace file does exist on entry to the utility (step (ii)); and the other is that the trace file does not exist on entry to the utility (step (iii)).

The Examiner is again correct to imply that between step (ii) when the trace file is created and step (iii) when the

Appln. No. 09/630,411
Amd. dated June 25, 2004
Reply to Office Action of March 25, 2004

trace file is analyzed, a debugger command is embedded in the trace file. But this may be done manually and will then not be part of the computer-implemented method. It is therefore respectfully urged that there is no need to recite this step explicitly and that it is sufficient to imply that it was carried out by the time step (iii) is carried out from the terminology of the claim which requires that the debugger be invoked "so as to execute a debugger command embedded in the trace file in place of the traced value." How or when it was embedded is immaterial so far as claim 7 is concerned and it is therefore submitted that the rejection under 35 U.S.C. §112, first paragraph is unwarranted.

35 U.S.C. §112, second paragraph

The rejected claims have been amended to eliminate the sources of indefiniteness noted by the Examiner. Claim has been amended to refer to "invocation of the utility", antecedent basis for which will be found at claim 1(c). Claim 5 has been amended to depend from claim 4, which provides antecedent basis for "the mechanism". Claim 6 has been amended to depend from claim 4, which provides antecedent basis for "the interleaving". Claim 12 has been amended to recite "during running of" a respective thread. Claim 15 has been amended to delete "the wrong time" and simply recite "stopping the program if said bipartite matching is not possible." This, of course, avoids execution of debugger commands.

Reconsideration and withdrawal of the rejections under 35 U.S.C. §112 is requested.

* * * * *

The rejections under 35 U.S.C. §102(e) and §103 are also respectfully traversed and reconsideration thereof is requested.

35 U.S.C. §102(e)

The Examiner rejected claims 1, 18, 19 and 24 as being anticipated by Poteat et al. (US 5,970,245). This rejection is respectfully traversed. Poteat et al. relates to a trace DLL that comprises a plurality of trace procedures having a one-to-one correspondence with the target procedures contained in a target DLL. Calls to target procedure in the target DLL may be monitored without disrupting the functionality of the target procedure or limiting access to the target DLL.

Poteat et al. is directed to the problem of allowing debugging of functions and procedures that are contained in DLL's (Dynamic Link Libraries) which are pre-compiled and so do not normally allow access to their internal code by a debugger. Poteat et al. overcome this problem by linking the main module to an intermediate DLL, which they refer to as a trace DLL and which, being created by, and therefore accessible to, the programmer who wants to debug the main module, can be modified with respect to the original DLL, referred to as the Target DLL. This allows statements in the Target DLL to be skipped and thereby allows, presumably by trial and error, for faulty code in the Target DLL to be isolated.

It is accepted that the trace DLL is a file. It also appears that Poteat et al. show in Fig. 3 a software

application (26) that invokes a call to a procedure called FUNCTION1 that is contained in a Trace DLL (28), which itself invokes a call to a function referred to as AEP_FUNCTION1 that is included in the Target DLL (30). As we understand the Examiner's reasoning, he considers AEP_FUNCTION1 to be a debugger command that is contained in a trace file, and he considers FUNCTION1 to be analogous to the utility recited in Claim 1 that reads the trace file 28. A modified form of FUNCTION1 containing the call to a different entry point of the same function is contained in the trace file, which is linked to the software application 26 so that control is passed to the trace file. AEP_FUNCTION1 is called by the trace file 28 whereby control is branched to the target DLL 30, which is linked to the trace file 28.

Claim 1 has been amended to clarify the distinctions of the invention over Poteat *et al.* Thus, in Poteat *et al.* FUNCTION1 is a "utility" that operates uniquely and always calls the procedure in whichever DLL is linked to the main module. Moreover, the mechanism described by Poteat *et al.* requires that FUNCTION1 calls the trace DLL and so cannot function if the trace file is absent. In contrast to this, the utility recited in claim 1 has two different modes of operation depending on whether or not the trace file exists. Thus, claim 1 now specifies that the utility is configured for creating a trace file if the trace file does not exist on entry to the utility; while if the trace file *does* exist on entry to the utility, then the utility reads the trace file and invokes at least one predetermined debugger command that has been previously embedded in the trace file at a desired location. A preferred mechanism for carrying this out is defined in claim 7.

By such means, the invention allows two-pass debugging of the program depending on whether the trace file exists or not. In the first pass, where the trace file does not yet exist, the utility creates the trace file and operates as a print (or write) statement that prints values of selected variables to the different lines of the trace file. The programmer can then cause automatic debugging of the program at the specific line in the program where a variable appears to be suspect by changing the corresponding line in the trace file to a debugger command. On re-running the program, the utility now sees that the trace file exists and so has no need to create it. Instead, it reads each line of the trace file where the utility is embedded and compares the corresponding value with the value of the variable at the respective line of the program. They should, of course, be identical except at a location where the programmer changed the entry in the trace file to a debugger command. Therefore, when reaching this line of the trace file and seeing that the value of the variable and the "value" appearing in the trace file are different, the utility construes the "value" in the trace file as a debugger command and invokes the debugger command automatically.

There is no disclosure, or even suggestion, in Poteat *et al.* to invoke a utility that creates a trace file if it does not exist on entry to the utility and otherwise reads the utility to read the trace file and invoke a debugger command. Therefore, it is respectfully submitted that claim 1 is patentable over Poteat *et al.*

Similar amendments have been effected to claims 18, 19 and 24, which are therefore submitted to be allowable for the same reasons.

35 U.S.C. §103

Claims 3 and 26 were rejected under 35 U.S.C. §103 in view of Poteat et al. in combination with Swoboda (US 6,388,533). It is respectfully submitted that the rejection is moot in view of the amendments to claims 1 and 24, respectively.

Claims 4-6 and 26 were rejected under 35 U.S.C. §103 in view of Poteat et al. in combination with "*Deterministic Replay of Java Multithreaded Applications*." It is respectfully submitted that the rejection is moot in view of the amendments to claims 1 and 24, respectively.

Claims 7-11, 20-23, 25 and 28 were rejected under 35 U.S.C. §103 in view of Poteat et al. in combination with "*Implicit-Specification Errors and Automatic, Trace-Based Debugging*" by Okie et al. The Examiner avers that Okie teaches checking whether a trace file exists and if the trace file does not exist on entry to the utility, creating the trace file and writing a traced value of at least one variable thereto at a desired location in the program. The Examiner supports this construction on Section 3 but we could find no suggestion in Okie to create a trace file conditionally, i.e. if it does not exist on entry to the utility. Rather, it is submitted that Okie describes a variation on conventional trace analysis where trace statements are added to the program being debugged which print statements to a trace file. The program is then executed and the trace file is analyzed.

Moreover, contrary to the Examiner's statement, we could find no suggestion in Section 5 of Okie to compare a current value of the least one variable with a respective line in the trace file and to invoke a debugger if they are different so as

to execute a debugger command embedded in the trace file in place of the traced value.

It is therefore respectfully submitted that claim 7 is patentable over Okie and Poteat *et al.*

If this rejection is repeated, it is asked that the Examiner identify more specifically the relevant disclosures of Okie.

All of the other claims were rejected in view of either Okie or Poteat *et al.* and it is therefore submitted that these too are patentable for the reasons stated above.

Supplemental comments

The algorithm as defined in claim 7 as originally drafted does not allow the utility to be invoked more than once in the same program. The reason for this is probably best seen with reference to claim 7 from which it emerges that there are related operations:

- i) creating the trace file;
- ii) writing to the trace file; and
- iii) reading the trace file so as to invoke the debugger command.

The way the claim was originally presented, writing the variables to the trace file was contingent on creating the trace file. But this is true only if a single variable only is to be tested. In practice, of course, it is desirable (albeit not mandatory) to write multiple trace variable to the trace file and then invoke the debugger so as to analyze the program where it appears that one or more variables may be suspect.

The original claim would not allow this because the test as to whether to write to or read from the trace file was conditional on whether the trace file exists on entry to the utility. Thus, if the utility is embedded in the calling program several times, the trace file would be created and the traced variable written only for the first invocation. On subsequent invocations, the trace file would be flagged as existing and so, according to the logic of the claim as previously presented, the trace file would be read. Therefore, what is actually required is that the test as to whether to write to or read from the trace file be conditional on whether the trace file exists on entry to the *program* and not the utility.

The relevant claims have therefore been corrected by way of introducing a *compare flag* that is set to one of two values depending on what is required. No new matter is added and the Examiner's attention is drawn to page 6, lines 11-16, stating:

When the program is executed on a test, a flag called "**compare**" is added. If **compare=false**, then the **Trace-Print** function prints Content into file **Trace-Name** in the form "**Trace: Content**".

If **compare=true**, then the program is executed under a modified debugger.

Claim 1 has been amended in a similar spirit but slightly differently. Thus in claim 1 there has been recited the concept of running the program twice, such that during a first pass the variables are written to the trace file; and during the second pass the trace file is read.

New claims 36 to 43 recite setting the compare flag to the second value on exit from the program so that on re-

Appln. No. 09/630,411
Amd. dated June 25, 2004
Reply to Office Action of March 25, 2004

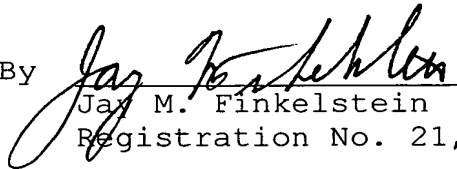
running the program, the read/compare mode of the utility will be invoked.

In view of the foregoing, it is requested that all of the objections and rejections of record be reconsidered and withdrawn, that the pending claims be allowed and that the application be found in allowable condition.

If the above amendment should not now place the application in condition for allowance, the Examiner is invited to call undersigned counsel to resolve any remaining issues.

Respectfully submitted,

BROWDY AND NEIMARK, P.L.L.C.
Attorneys for Applicant

By 
Jay M. Finkelstein
Registration No. 21,082

JMF:dtb
Telephone No.: (202) 628-5197
Facsimile No.: (202) 737-3528
G:\BN\C\cohn\Farchil\pto\AMD 24JUN04.doc